

12-20-99

A

FISH & RICHARDSON P.C.

2200 Sand Hill Road
Suite 100
Menlo Park, California
94025

Telephone
650 322-5070

Facsimile
650 854-0875

Web Site
www.fr.com

December 17, 1999

Attorney Docket No.: 07844-280001

Box Patent Application

Assistant Commissioner for Patents
Washington, DC 20231

Presented for filing is a new original patent application of:

Applicant: ROBERT J. CHANSLER

Title: USER INTERFACE SUBSTITUTION

Enclosed are the following papers, including those required to receive a filing date
under 37 CFR 1.53(b):

	<u>Pages</u>
Specification	14
Claims	6
Abstract	1
Declaration	1
Drawing(s)	8

Enclosures:

- Assignment cover sheet and an assignment, 3 pages, and a separate \$40 fee.
- Appendix (15 pages)
- Postcard.

Basic filing fee	\$760
Total claims in excess of 20 times \$18	\$270
Independent claims in excess of 3 times \$78	\$156
Fee for multiple dependent claims	\$0
Total filing fee:	\$1186

A check for the filing fee is enclosed. Please apply any other required fees or any credits to deposit account 06-1050, referencing the attorney docket number shown above.

If this application is found to be incomplete, or if a telephone conference would otherwise be helpful, please call the undersigned at (650) 322-5070.

Express Mail Label No. EL444262519US

December 17, 1999

Date of Deposit

12/17/99



Frederick P. Fish
1855-1930
W.K. Richardson
1859-1951

BOSTON

DELAWARE

NEW YORK

SILICON VALLEY

SOUTHERN CALIFORNIA

TWIN CITIES

WASHINGTON, DC

09/467310-121799



FISH & RICHARDSON P.C.

Assistant Commissioner for Patents

December 17, 1999

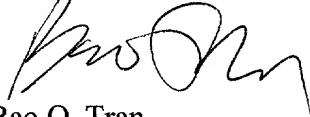
Page 2

Kindly acknowledge receipt of this application by returning the enclosed postcard.

Please send all correspondence to:

Customer Number 021876

Respectfully submitted,



Bao Q. Tran

Reg. No. 37,955

Enclosures

BQT/clp

50003528.doc

09467310-121799

APPLICATION
FOR
UNITED STATES LETTERS PATENT

TITLE: USER INTERFACE SUBSTITUTION

APPLICANT: ROBERT J. CHANSLER

Express Mail Label No. EL444262519US

December 17, 1999
Date of Deposit

66444262519US

USER INTERFACE SUBSTITUTION

BACKGROUND

The invention relates to systems and methods of defining a user interface for computer program.

5 Traditionally, graphical user interfaces have been employed to facilitate interaction of a user with software in a computer system. For example, typical objects in a GUI are icons presented on a background or "desktop", with the icons representing a set of functions, application programs, system resources, or the like. These icons may be selected to instruct the computer to perform particular functions.

10 GUI objects are generally created as part of a static application program construct that describes how a particular GUI object appears and what function is performed when that GUI is selected. Many visually-based application builder tools, such as Visual Basic, available from Microsoft Corporation of Redmond, Washington, have a graphical user interface that includes a "toolbox", a "form window", and a "property sheet," among others. A toolbox
15 typically contains icons that represent different classes of components, or software modules. During a UI (user interface) construction session, a form window is displayed for the user to compose an application UI. A component selected from the toolbox may be "dropped", or otherwise placed, within the form window. Placing a component means placing an icon or other visual representation of the component, within the form window. A property sheet
20 displays properties, or attributes, which relate to the selected component. The properties can include, but are not limited to, information relating to the size, color, and name of the

selected component. Programmers then insert code for each object to handle events generated by a user during operation.

After an application has been developed, a user generally has a limited ability to customize the user interface by modifying properties associated with certain GUI objects. In general, a property sheet displays a fixed set of properties that can be modified to customize, or configure, a given component. Hence, a user can select and modify different properties in the property sheet in order to customize a component. Simple components typically have fewer associated properties than larger components, and can therefore be readily customized by making selections in a property sheet.

As computer usage becomes increasingly widespread, the desire for custom computer applications that are specific to the needs of a particular user is also increasing. The desire is driven in part by the need to customize the software to the user's needs rather than force the user to adjust to the software, which can be tedious, time-consuming and non-productive.

SUMMARY OF THE INVENTION

In one aspect, a method defines a user interface for a computer program after execution of the computer program has begun by: reading a function description of a first function to be provided by the user interface; reading an appearance description of a first appearance to be presented by the user interface; associating the function description and the appearance description on the fly at run time; and executing the user interface with the associated function and appearance.

Implementations of the invention may include one or more of the following. The function description or the appearance description can be replaced during program execution. The method can include reading a map defining multiple functions to be provided by the user interface including the first function; reading a fashion defining all appearances to be

presented by the user interface including the first appearance; associating the map and the function on the fly at run time; and executing the user interface with the associated map and function. The map or fashion can be replaced during program execution. The map can specify that a subordinate part of the user interface is specified by a second map-fashion pair.

- 5 Events can be received from the map component or the fashion component. Business logic associated with the respective component can be executed. The components can be stored in a database.

In a second aspect, a method defines a user interface for a computer program by:
associating a map component and a fashion component on the fly at run time to generate the
10 user interface; and executing the user interface with the associated function and appearance.

Implementation of the method can include one or more of the following. A resource bundle associated with the map component can be loaded. Sub-components of the user interface can be loaded and instantiated. The fashion component can be called to allocate a resource to each sub-component. Each sub-component can be instructed to present itself in
15 the user-interface. Events from the map component or the fashion component can be received.

Advantages of the invention include one or more of the following. The map (controller) and the fashion (view) need not be static and each can be changed (or can change itself) independently of the other during execution of a program. By changing the map, the
20 logic of the user interface can change while the same fashion is used to provide continuity of presentation. By changing the fashion without changing the map, the logic of the user interface can be presented with radically different appearances. Further, to provide

flexibility, the map can specify that a subordinate part of a user interface is to be constructed using a second map-fashion pair.

The logic and appearance of a user interface can be built independently, making the UI presentation more accessible to non-programmers. The UI can be modified without re-
5 compiling the application. Because of late binding, the details of the UI presentation are established at execution time for the application. Hence, user interface development can be decoupled from code development. Hence, a visual designer can design a user interface without incurring delays associated with involving a programmer.

The UI specification is declarative rather than procedural. The declarative aspect is
10 advantageous in that abstract, high-level specifications can be synthesized without a detailed procedural description of the required action, such as calling the system to allocate a particular resource, to get a layout object, to modify the layout object, and to attach the modified layout object to another layout object.

The appearances can be presented using a graphical interface, or can be presented in a
15 non-visual modality such as a voice user interface that can be used in speech driven computers, including telephones and voice activated computers. Moreover, much of the UI is exposed in a way that facilitates localization. Thus, it is easy to create a user interface for an international product in which customers in different see different text, images, and other user interface elements. Use of the invention allows a corporation or other group to
20 implement a corporate appearance for its user interface across multiple products.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing a system with an exemplary user interface object.

Fig. 2 is a diagram illustrating an exemplary user interface.

Fig. 3 is a diagram illustrating cascaded user interface objects.

Fig. 4 is a flowchart illustrating a process for generating presentations based on the user interface objects.

Fig. 5 is a flowchart illustrating the initialization of objects associated with a user
5 interface.

Fig. 6 is a flowchart illustrating the operation of the user interface initialized in Fig. 5.

Fig. 7 is an exemplary representation of the specification of a user interface.

Fig. 8 is a diagram illustrating an embodiment of a process to generate a user
interface presentation.

Fig. 9 is a flowchart illustrating a second process for generating presentations based
10 on the user interface objects.

Fig. 10 is a diagram of an operating environment for the process of Fig. 4 or Fig. 9.

Fig. 11 is a diagram of an exemplary computer capable of executing the process of
Fig. 4 or Fig. 9.

15 **DETAILED DESCRIPTION**

As shown in Fig. 1, a system 100 has a user interface (UI) object. The UI object 102 can have a graphical modality such as a two-dimensional or three-dimensional graphical user interface, or can have a different modality, such as a voice user interface commonly used in telephone response system.

20 The object 102 has a map component 104. The map component 104 contains instructions for handling and processing objects contained in the user interface. The map component 104 thus embodies logic associated with the user interface object 102. The map component 104 knows that certain UI resources are available to it and are defined in a

resource bundle 106 with a list of UI objects and their characteristics. The resource bundle can be created by an application programmer or can be programmatically generated. The resource bundle contains, directly or by reference, the layout rule for each UI object and the requisite system call that displays and enable that UI object to generate an event based on a user input.

The UI objects in the resource bundle 106 can include, for example, a text-box that provides an area to enter or display text. The UI object can be a checkbox, which displays a True/False or Yes/No option. The UI objects can include combo-box that combines a text box with a list box and allows a user to type in a selection or select an item from a drop-down list. The UI objects can include a button that carries out a command or action when a user chooses it. Yet another exemplary UI object can be a horizontal scrollbar and vertical scrollbar, which allow a user to add scroll bars to controls that do not automatically provide them.

Resource bundles contain locale-specific objects. When the application software needs a locale-specific resource, a "String" for example, the program can load it from the resource bundle that is appropriate for the current user's locale. In this way, program code can be largely independent of the user's locale by isolating most, if not all, of the locale-specific information in resource bundles. Each related subclass of the resource bundle contains the same items, but the items have been translated for the locale represented by that resource bundle subclass. For example, both MyResources (English version) and MyResources_de (German version) may have a String that is used on a button for confirming operations. In MyResources the String may contain "OK" and in MyResources_de it may contain "Gut."

When the program needs a locale-specific object, it loads the resource bundle class using a "getBundle()" method. The resource bundle lookup searches for classes with various suffixes on the basis of (1) the desired locale and (2) the default locale (baseclass); the search order is from lower-level (more specific) to parent-level (less specific). The result of the
5 lookup is a class, but that class may be backed by a property file on disk. If a lookup fails, getBundle() throws a missing resource exception. Such an exception will generally cause the map component 104 to rely on a default locale.

The map component 104 contains logic to handle one or more events generated by the UI objects. An event has a life cycle that begins with the action or condition that initiates the
10 event and ends with the final response by the event handler. Typically, a user action or condition associated with the event occurs, and an event object is updated to reflect the conditions of the event. The event fires to notify the logic in the map component 104. The map component logic executes an event handler associated with the event is called. The event handler carries out its actions and returns. The event is provided to the next element in
15 the hierarchy, and the event handler for that element is called. For events that bubble, if there is no event handler bound to the source element generating the event, the event handler for the next element up the element's hierarchy is called.

The user interface object 102 also contains a fashion component 108. The fashion component 108 embodies the presentation, including the layout of the object 102. The
20 fashion component is a collection whose elements represent each loaded form in an application. The fashion component 108 also derives its knowledge from a resource bundle 110. In one embodiment, the resource bundle 106 is the same as the resource bundle 110.

The user interface object 102 also can include additional user interfaces 110, 112 and 114. The UIs 110, 112, and 114 provide programmatical support for the user to navigate to

another presentation. In one embodiment, the UI 110, 112 or 114 can be a button which, when clicked, moves users backwards or forwards in the application.

During initialization, the application associated with the user interface object 102 loads the map component 104 and the fashion component 108 before creating a presentation to the user. The map component 104 and the fashion component 108 need not be static. The map component 104 and the fashion component 108 can be replaced during execution. By changing the map component 104, the logic of the user interface can be altered while the same fashion component 108 can still be used to provide a continuity of presentation.

Correspondingly, the fashion component 108 can be changed during execution to allow the logic of the user interface to be presented with a radically different appearance or even in some non-visual modality, such as the voice user interface.

A variety of data components can be identified with one or more map components and one or more fashion components. The map component as well as the fashion component can change under the control of their code or can be changed by other map and fashion components. Thus, the user interface object 102 of Fig. 1 allows for an arbitrary composition. Two parts of the application user interface can be built independently using different map components or fashion component pairs. Alternatively, one map component can specify that a subordinate part of the user interface be constructed by a second map component 104 and a second fashion component 108 pair as needed.

Fig. 2 illustrates an exemplary user interface. The user interface of Fig. 2 includes a top horizontal portion 120 that is positioned above a left lower portion 122 and right lower position 124. Further, each of the interface portions 120, 122 and 124 in turn can reference one or more secondary user interfaces.

Fig. 3 shows an exemplary structure associated with the user interface embodiment of Fig. 2. A user interface object 128 includes a map component 130 and a fashion component 132. The map component 130 and fashion component 132 each references one or more resource bundles that specify the UI objects such as buttons and text-boxes, for example. Moreover, the user interface 128 references a top user interface object 134, a right user interface object 136 and a left user interface object 138. During execution, the logic associated with the map component 130 finds the required subcomponents. In this case, the logic finds three objects 134, 136 and 138. The logic instantiates each object using the name of the map and the fashion. The user can navigate down the user interface hierarchy for subsequent display screens by selecting references to dependent user interface objects using various UI objects.

The exemplary structure of Fig. 3 also shows details associated with the left user interface object 138. Similar to the parent user interface object 128, the left user interface object 138 can designate a plurality of additional user interface objects 140, 142, 144, and 146. Although Fig. 3 only shows UI objects associated with the left user interface objects 138, it is to be understood that similar objects exist for the top and right user interface objects 134 and 136. Generally, the UI is developed through a process of recursive refinement until primitive UI components (a button or a text label, for instance) are specified. The resulting structure is a rooted, directed graph. Also, the same map or fashion may appear at different levels of the UI composition. More particularly, for two component UI objects (each of class UI) with map and fashion objects, the two maps (fashions) are different objects but may be the same class. For example, in the actual application, certain classes for simple navigation and for providing a particular look and feel for conventional UI widgets are used repeatedly.

Turning now to Fig. 4, a process associated with loading and using the user interface is shown. First, the map component and a fashion component are loaded (step 202). Next, a user interface is generated based on the map component and the fashion component loaded in step 202. A user interface or presentation is then presented to the user (step 204). Next, the application determines whether a new presentation is required (step 206). If so, the map and fashion components are updated (step 208) and the process 200 loops back to step 204 to present the user interface to the user. Alternatively, if a new presentation is not required, the process 200 executes the logic of the application (step 210).

After executing the business logic, the process 200 determines whether additional presentations are required (step 212). If so, the process 200 loops back to step 208 to reference new map and fashion components (step 208). Alternatively, if a new presentation is not required in step 212, the process 200 determines whether the user wishes to exit the application (step 214). If not, the process loops back to step 210 to continue executing the application. From step 214, if the user wishes to exit, the process 200 exits.

Fig. 5 shows a process 230 for initializing a user interface. First, the process loads one or more resource bundles associated with the map component or with a fashion component. Next, each subcomponent of the user interface is located (step 234). The located subcomponents are then instantiated (step 236). Next, the process 230 calls the fashion component to allocate resources to the subcomponents (step 238). Finally, each component is then instructed to present itself on the screen as a user interface element or object (step 240).

Fig. 6 illustrates the operation of the map component 104. In the process 250, the map component 104 instructs the fashion component 108 to present itself (step 252). Next,

the map component 104 listens for events generated during the presentation or by the fashion component 108 (step 254). The logic of the application is then executed (step 256).

Fig. 7A shows pseudo-code associated with an exemplary user interface “next.” The “next” user interface includes a plurality of maps: Map1-Map3. Map 1 is associated with fashion 1 which defines a user interface region called top. Map 2 is associated with a fashion component 2 and is associated with a right region. Finally, Map 3 is associated with fashion component 3 and is used to define a left region.

Pseudo-code associated with the user interface of Fig. 2 is shown in Fig. 7B. In this figure, the layout of the user interface is defined as having a top region such as the region 120 of Fig. 2 as a vertical region. The vertical region is further specified as having a right region such as the region 124 of Fig. 2 and a left region such as the region 122 of Fig. 2, both of which are laid horizontally adjacent each other under the top region.

Fig. 8 is a diagram 300 illustrating the operation of one implementation embodying the map and fashion components. First, a user interface is created (step 301). The user interface includes a map component 304, a fashion component 306 and one or more properties files 308. Before the user interface is executed, a binding process 302 combines the map fashion and property files into an executable ready form. During operation, the process 300 uses the fashion components to interpret the property files (step 310).

Additionally, the map components can be used to interpret various selection events generated by the user (step 312). Further, the map components can be used to interpret the property files (step 316).

During operation, the application can query a database, which can be resident with a host application or can be accessed remotely. In response, the database can provide information indicating the availability of different map components and fashion components.

In one implementation, the identification of a server that will handle the query is provided in the query. Based on the response, an initial selection of the map component 104 and the fashion component 108 is made based on the query response. The selection can also be based on locale, which customizes the selection base on the geographic location of the user.

5 Based on the response to the query, an initial choice is made according to the environment variables and established preferences. A initial user interface is then presented to the user. The user can make certain selections using the UI defined in the map component or fashion component, and the presented interface can then be altered accordingly. Additionally, program logic embedded in the map component 104 can dictate the change of
10 the map component 104 or the fashion component 108, respectively.

 Fig. 9 shows a process 320 that is executed in one embodiment of the invention is shown. First, an application program is initiated (step 322). The execution of the program generates one or more queries directed at a database enquiring as to the availability of maps, fashions and locale (step 324). Next, a responsive database generates various environment
15 variables and preferences, which are sent back to the application program (step 326). Next, the application program selects a map component (step 328). The application program then selects and loads a fashion component (step 330). Finally, a locale is selected and loaded (step 332). Next the user interface is created user (step 324) and presented to the user (step 336). During this presentation, the fashion component uses the locale to localize data
20 presented to the user (step 338). Further, the program and/or business logic information contained in the map component is used to either change the map component itself or the fashion component (step 340). From step 336, the user can choose another map, fashion or locale (step 342).

Fig. 10 shows apparatus associated with the process of Fig. 9. First, a remote database containing map components, fashion components and locale information is connected to a network 410. In addition to the map, fashion and locale information in the database 402, a property file can be contained in a second database 404 that is connected to the network 410.

5 On the other side of the network, the request is handled by application software, operating system interfaces and device drivers 422. The application, operating system interfaces and device drivers receive user instructions, as captured by an input/output hardware interface 424. The hardware 424 in turn drives a display device 426 and receives inputs from a user input device 428. Based on the map component, fashion component and
10 locale information, an initial user interface is generated. The user interface can then be presented based on the user input as provided to the user input device 428. For each presentation, the map and the fashion can be changed independently of the other during execution of a program.

As shown above, the logic and appearance of a user interface can be built
15 independently. Hence, an application visual designer can create the user interface without incurring delays associated with involving a programmer. Moreover, the UI specification is declarative rather than procedural. Thus, high-level specifications can be synthesized without a detailed procedural description of the required action. The appearances can be presented using a graphical interface, or can be presented in a non-visual modality such as a
20 voice user interface that can be used in speech driven computers, including telephones and voice activated computers. Moreover, much of the UI is exposed in a way that facilitates localization by presenting localized text, images, and other user interface elements.

The techniques described here may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs

executing on programmable computers that each includes a processor, a storage medium readable by the processor (including volatile and nonvolatile memory and/or storage elements), and suitable input and output devices. Program code is applied to data entered using an input device to perform the functions described and to generate output information.

- 5 The output information is applied to one or more output devices.

Figure 11 illustrates one such computer system 600, including a CPU 610, a RAM 620, and an I/O controller 630 coupled by a CPU bus 640. The I/O controller 630 is also coupled by an I/O bus 650 to input devices such as a keyboard 660 and a mouse 670, and output devices such as a monitor 680. Variations are within the scope of the following

10 claims. For example, instead of using a mouse as the input devices, a pressure-sensitive pen or tablet may be used to generate the cursor position information.

Moreover, each program is preferably implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case,

15 the language may be a compiled or interpreted language. Each such computer program is preferably stored on a storage medium or device (e.g., CD-ROM, hard disk or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described. The system also may be implemented as a

20 computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

While the invention has been shown and described with reference to an embodiment thereof, those skilled in the art will understand that the above and other changes in form and detail may be made without departing from the spirit and scope of the following claims.

What is claimed is:

1. A method of defining a user interface for a computer program, comprising:
after execution of the computer program has begun, defining a user interface
of the program by:

5 reading a function description of a first function to be provided by the user
interface;

reading an appearance description of a first appearance to be presented by the
user interface;

10 associating the function description and the appearance description on the fly
at run time; and

executing the user interface with the associated function and appearance.

2. The method of claim 1, further comprising replacing the function description
during program execution.

15 3. The method of claim 1, further comprising replacing the appearance
description during program execution.

4. The method of claim 1, further comprising:
reading a map defining multiple functions to be provided by the user interface
including the first function;

20 reading a fashion defining all appearances to be presented by the user
interface including the first appearance;

associating the map and the function on the fly at run time; and

executing the user interface with the associated map and function.

5. The method of claim 1, further comprising replacing the map during program execution.

5 6. The method of claim 1, further comprising replacing the fashion during program execution.

7. The method of claim 1, wherein the map specifies that a subordinate part of the user interface is specified by a second map-fashion pair.

8. The method of claim 1, further comprising receiving events from the map component or the fashion component.

9. The method of claim 8, further comprising executing business logic associated with the respective component

10. The method of claim 1, wherein the components are stored in a database.

11. A method of defining a user interface for a computer program, comprising:

15 associating a map component and a fashion component on the fly at run time to generate the user interface; and

executing the user interface with the associated function and appearance.

12. The method of claim 11, further comprising loading a resource bundle associated with the map component.

13. The method of claim 12, further comprising locating sub-components of the user interface.

5 14. The method of claim 12, further comprising instantiating one or more sub-components of the user interface.

15. The method of claim 12, further comprising calling the fashion component to allocate a resource to each sub-component.

10 16. The method of claim 15, further comprising instructing each sub-component to present itself in the user-interface.

17. The method of claim 11, further comprising receiving events from the map component.

18. The method of claim 11, further comprising receiving events from the fashion component.

15 19. The method of claim 11, further comprising executing business logic associated with the map component

20. The method of claim 11, wherein the components are stored in a database.

21. Computer-readable medium to define a user interface for a computer program after execution of the computer program has begun, comprising instructions to:

read a function description of a first function to be provided by the user interface;

read an appearance description of a first appearance to be presented by the user interface;

5 associate the function description and the appearance description on the fly at run time; and

execute the user interface with the associated function and appearance.

22. The computer-readable medium of claim 21, further comprising instructions to replace the function description during program execution.

10 23. The computer-readable medium of claim 21, further comprising instructions to replace the appearance description during program execution.

24. The computer-readable medium of claim 21, further comprising instructions to:

15 read a map defining multiple functions to be provided by the user interface including the first function;

read a fashion defining all appearances to be presented by the user interface including the first appearance;

associate the map and the function on the fly at run time; and

executing the user interface with the associated map and function.

25. The computer-readable medium of claim 21, further comprising instructions to replace the map during program execution.

26. The computer-readable medium of claim 21, further comprising instructions to replace the fashion during program execution.

5 27. The computer-readable medium of claim 21, wherein the map specifies that a subordinate part of the user interface is specified by a second map-fashion pair.

28. The computer-readable medium of claim 21, further comprising instructions to receive events from the map component or the fashion component.

29. The computer-readable medium of claim 28, further comprising instructions
10 to execute business logic associated with the respective component

30. The computer-readable medium of claim 21, wherein the components are stored in a database.

31. A computer-readable medium of defining a user interface for a computer program, comprising instructions to:

15 associate a map component and a fashion component on the fly at run time to generate the user interface; and

execute the user interface with the associated function and appearance.

32. A system to define a user interface for a computer program, comprising:

a processor;

a device coupled to the processor to present the user interface;

means for associating a map component and a fashion component on the fly at

5 run time to generate the user interface; and

means for executing the user interface with the associated function and

appearance.

33. The system of claim 32, wherein the device is a display.

34. The system of claim 32, wherein the device is a sound input-output device.

10 35. The system of claim 32, wherein the device is a telephone.

ABSTRACT OF THE DISCLOSURE

Methods and apparatus define a user interface for a computer program after execution of the computer program has begun. A user interface for the program is defined by:
associating a map component and a fashion component on the fly at run time to generate the
5 user interface; and executing the user interface with the associated function and appearance.

50000834.doc

09467310-12199
50000834.doc

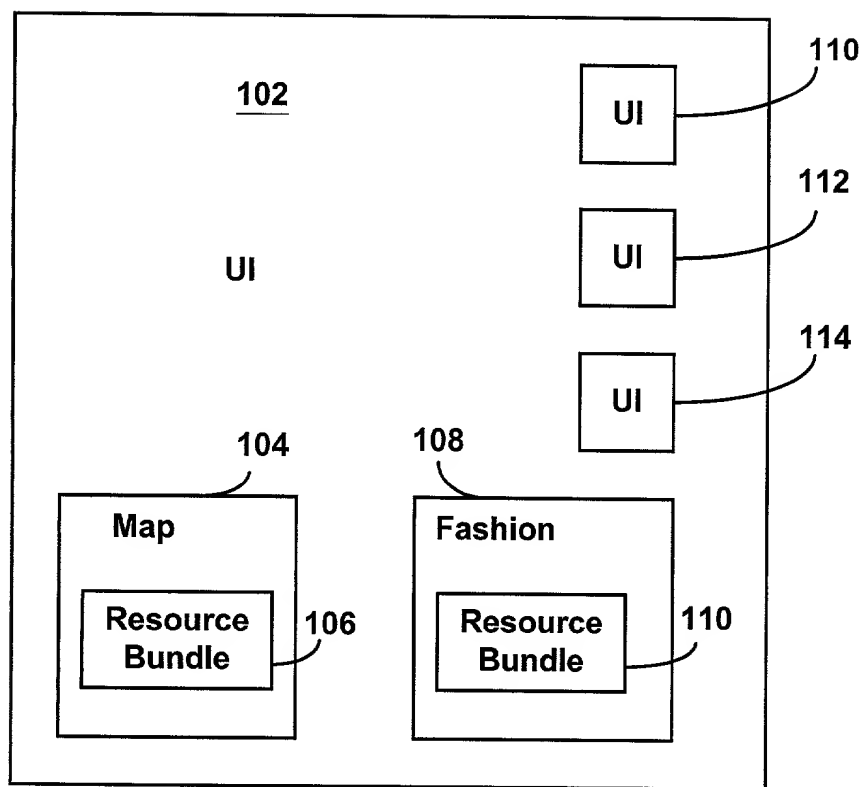


FIG. 1

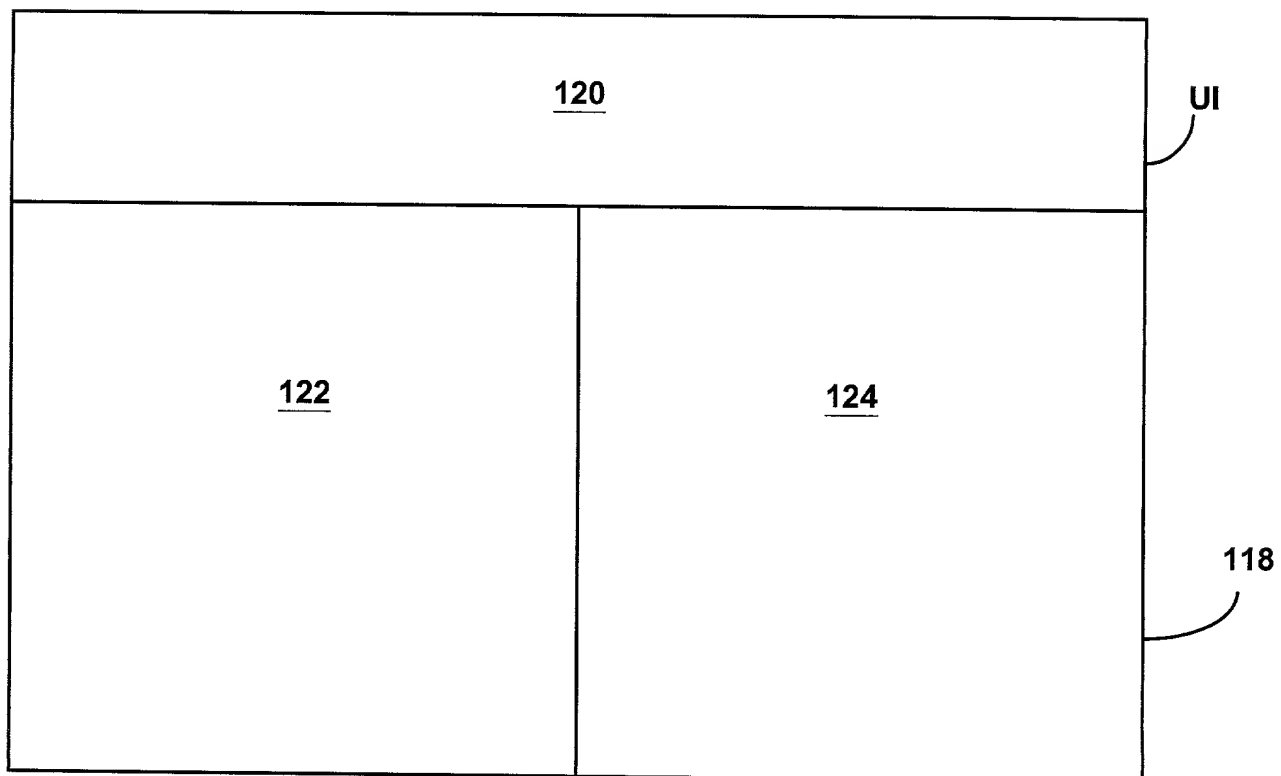


FIG. 2

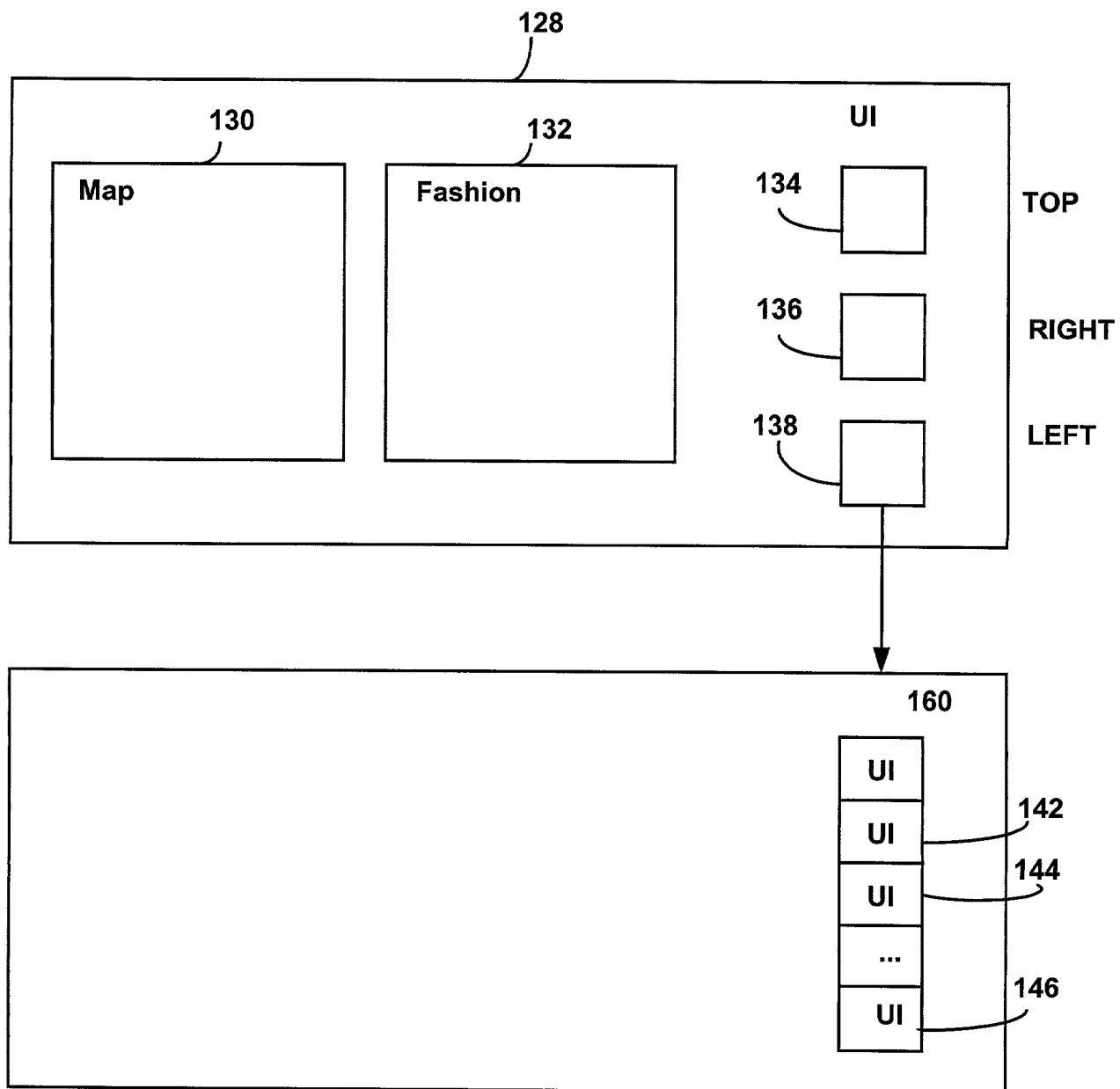


FIG. 3

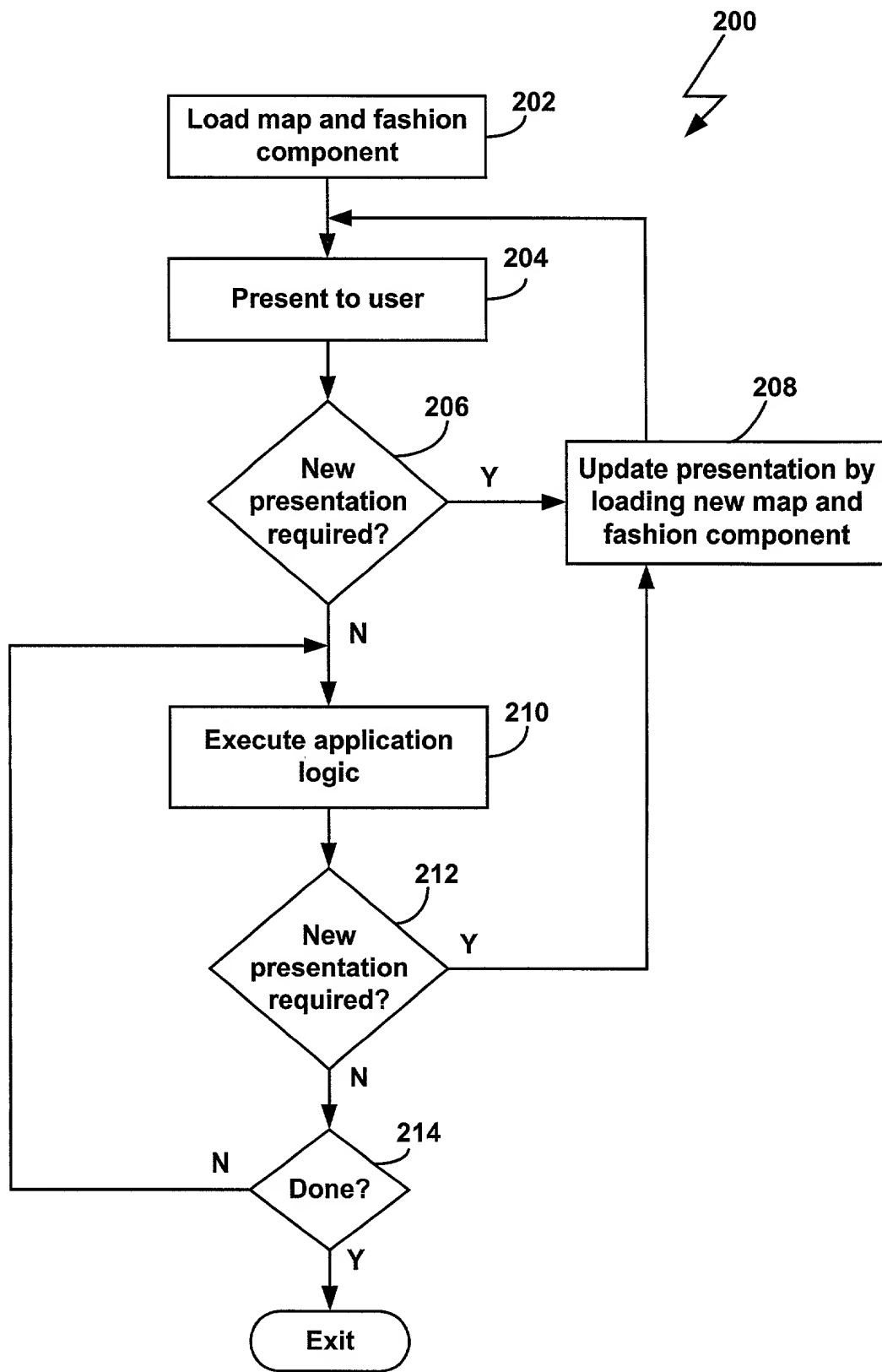


FIG. 4

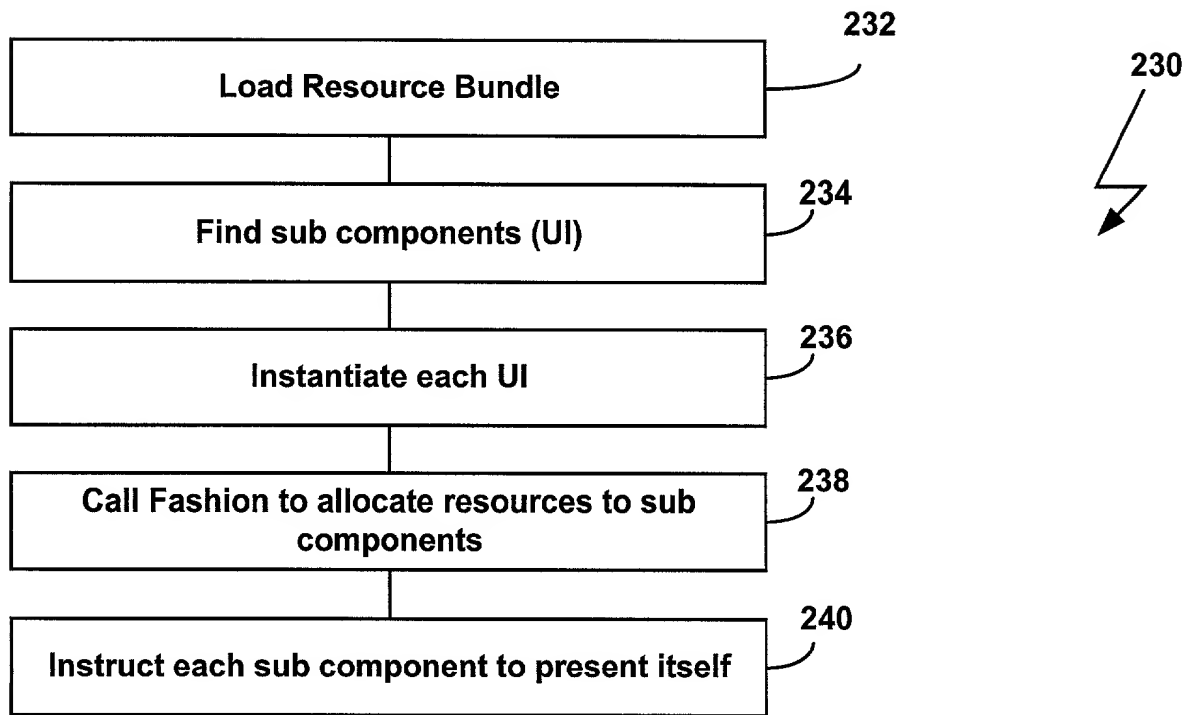


FIG. 5

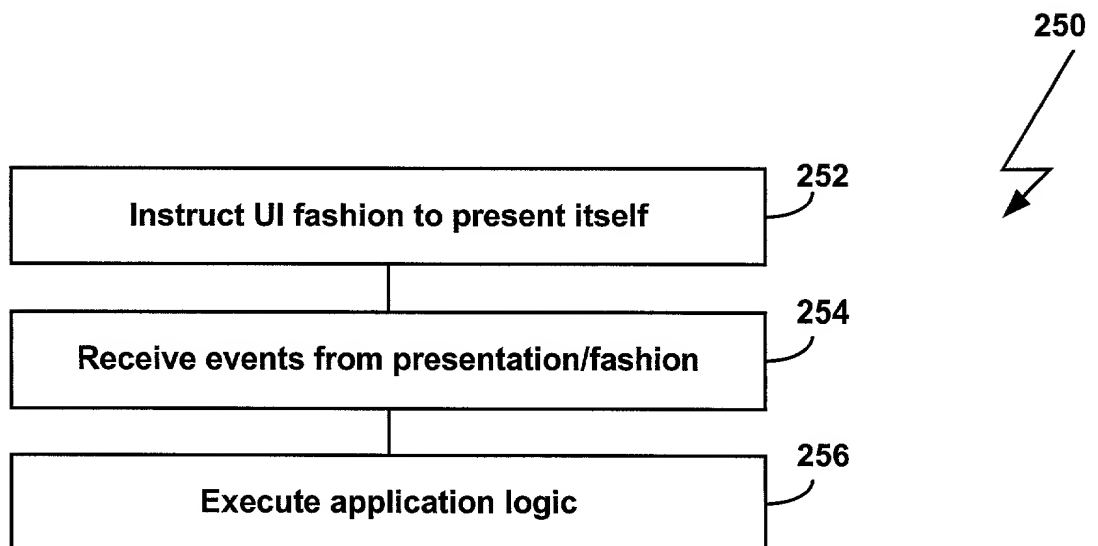


FIG. 6

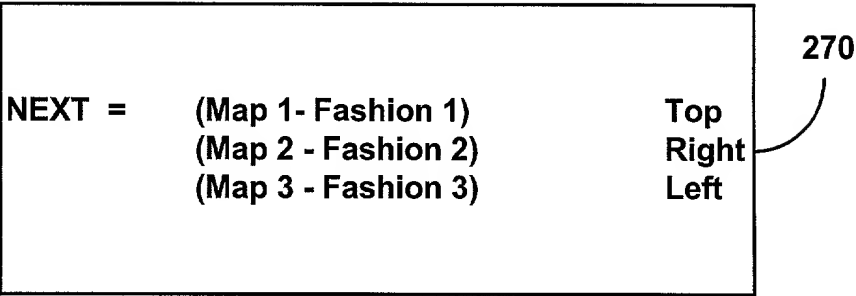


FIG. 7A



FIG. 7B

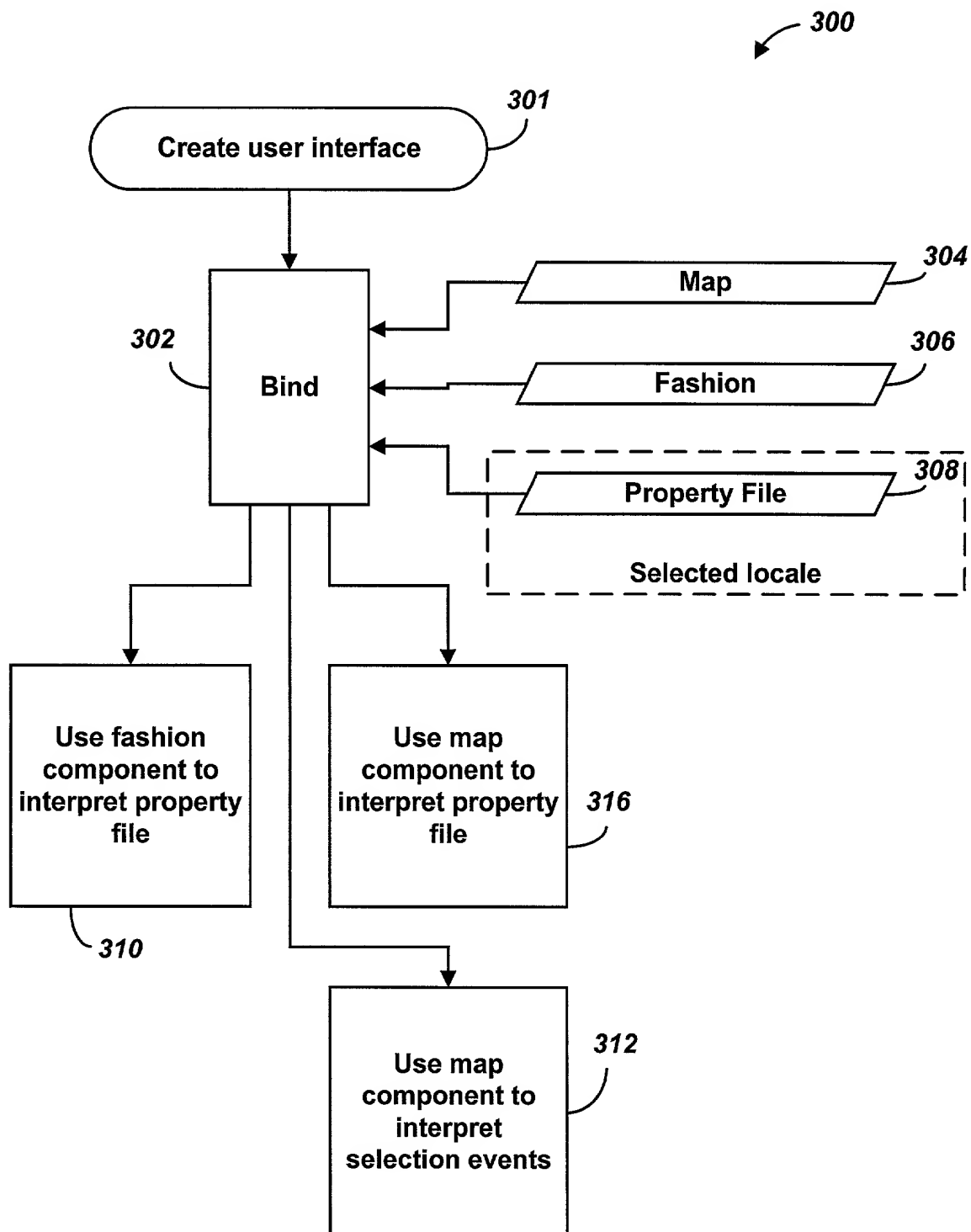


FIG. 8

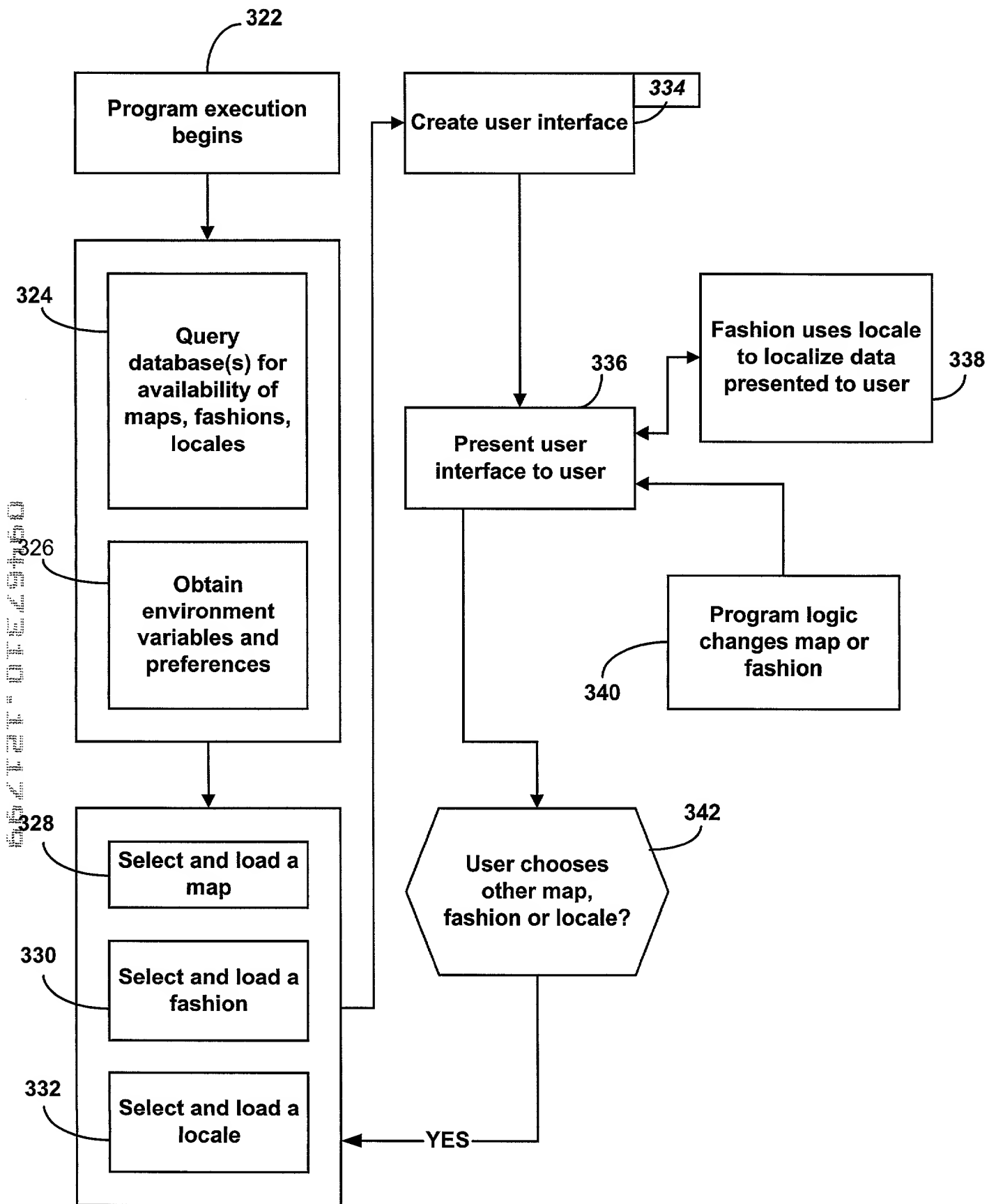


FIG. 9

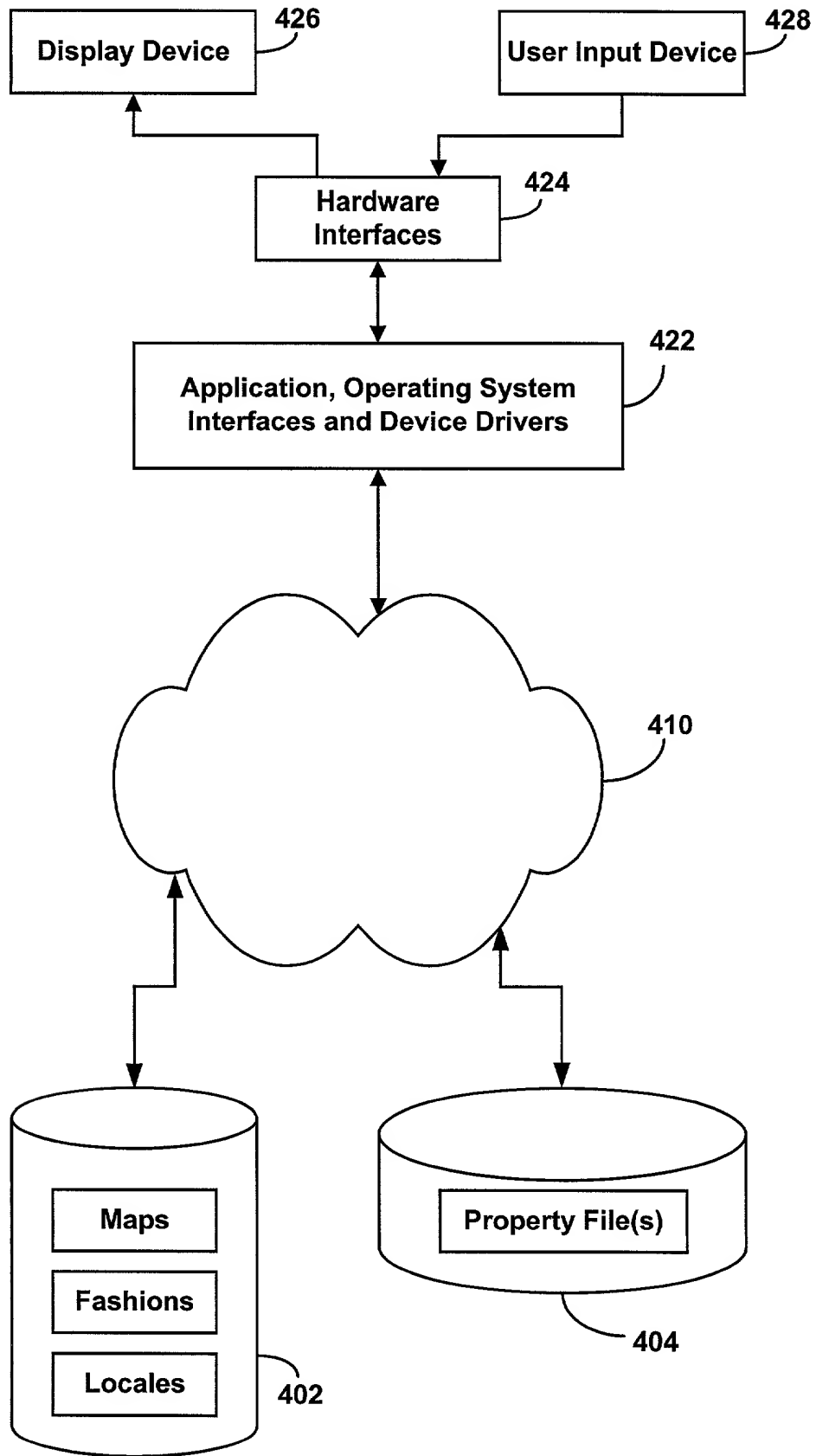


FIG. 10

COMBINED DECLARATION AND POWER OF ATTORNEY

Attorney Docket No. 07844/280001

Client No. P254

COMBINED DECLARATION AND POWER OF ATTORNEY

As a below-named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

USER INTERFACE SUBSTITUTION

the specification of which:

☒ is attached hereto.

☐ was filed on _____.

☐ under Application No. _____.

☐ with Express Mail No. _____ (Application Number not yet known).

☐ was described and claimed in PCT International Application No. _____.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, code of Federal Regulations, Section 1.56(a).

I hereby appoint the following attorneys and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith, and direct that all correspondence be addressed to that Customer Number:

Customer Number 021876

Address all telephone calls to **Bao Q. Tran., Reg. No. 37,955** at telephone number (617) 542-5070.

☒ For Assigned Inventions: I understand that the purpose of making this appointment is to permit prosecution of patent applications for the above-identified invention for the benefit of my assignee, and that this appointment does not create an attorney-client relationship between me and these appointees.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of inventor: Robert J. Chansler

Inventor's signature

Date:

Residence:

Citizen of:

Post Office Address:

Los Altos, California 94020

U.S.A.

660 Palm Avenue, Los Altos, California 94020

17 Dec 1999
Los Altos Hills, CA 94022
25700 Deertfield Dr
Los Altos Hills, CA 94022

09467310-131799

Attorney Docket No. 07844/28001
Client No. P254

APPENDIX

07844-28001-P254

APPENDIX

This appendix contains exemplary implementations of a system and a method for user interface substitution using Java. First, one exemplary Java code for a map base class is shown below:

```
/**
 * Interface implemented by each Map class.
 * <p>
 * This interface will grow as new interactions between Maps and Fashions
 * are defined.
 *
 * @version    %I%, %G%
 * @author    Chansler
 */
interface Map {
    /**
     * set the UI for this Map
     *
     * @param ui UI that created this Map
     */
    void setUI(UI ui);
    /**
     * get the UI for this Map
     *
     * @return UI Ui that created this Map
     */
    UI getUI();
    /**
     * Notify Map that the parent UI is quitting.
     */
    void dispose();
    /**
     * Notify Map that an action occurred in a choice tree.
     * This is a direct call from the Fashion.
     * (may have been more stylish to have used event registration.)
     * Clues are analyzed, and a dispatch is performed. If the action
     * cannot be completed locally, Preferences are set, and the UI is
     * notified. The UI is expected to generate a UIEvent and the Main
     * will conditionally restart the UI.
     *
     * @param group id of choice group
     * @param item id of chosen item from group
     * @param clue clue that identifies kind of choice group
     */
}
```

```

    * @param selected indicates state of bi-stable choice item
    */
    void treeAction(String group, String item,
                    String clue, Boolean selected);
    /**
     * Navigate to the ultimate beginning
     */
    void begin();
    /**
     * Navigate to the ultimate end
     */
    void end();
    /**
     * Navigate to the local beginning
     */
    void first();
    /**
     * Navigate to the local end
     */
    void last();
    /**
     * Navigate to the local previous
     */
    void previous();
    /**
     * Navigate to the local next
     */
    void next();
    /**
     * Navigate by name
     *
     * @param where name of UI to go to
     * @param first do a "first" when you get there
     * @param last do a "last" when you get there
     */
    void navigate(String where, Boolean first, Boolean last);
}

```

Exemplary Java code for a fashion base class is shown below:

```

/**
 * Interface implemented by each Fashion class.
 * <p>
 * This interface will grow as new interactions between Maps and Fashions
 * are defined.

```

```

*
* @version   %I%, %G%
* @author    Chansler
*/
interface Fashion {
    /**
     * set the UI for this Fashion
     *
     * @param ui UI that created this Fashion
     */
    void setUI(UI ui);
    /**
     * get the UI for this Fashion
     *
     * @return UI Ui that created this Fashion
     */
    UI getUI();
    /**
     * Notify Fashion that the parent UI is quitting.
     */
    void dispose();
    /**
     * Get subordinate container, if any.
     */
    java.awt.Container nextContainer();
    /**
     * Get subordinate container, if any.
     *
     * @param key id to select in Fashion's layout manager
     */
    java.awt.Container nextContainer(String key);
    /**
     * Get subordinate container, if any, from the i'th parallel display.
     *
     * @param i selects the i'th parallel display
     * @param key id to select in Fashion's layout manager
     */
    java.awt.Container nextContainer(int i, String key);
    /**
     * present a choice tree.
     *
     * @param id key of root description
     *
     * @return Object handle for choice tree
     */
    Object tree(String id);
}

```

```

/**
 * present a message to the user
 *
 * @param group    choice group
 * @param item     item from group identifying message
 */
void message(String group, String item);
/**
 * present a message to the user
 *
 * @param group    choice group
 * @param item     item from group identifying message
 * @param tail     extra run-time information
 */
void message(String group, String item, String tail);
/**
 * request confirmation of choice from the user
 *
 * @param group    choice group
 * @param item     item from group identifying message
 *
 * @return Boolean whether the user confirmed choice
 */
Boolean confirm(String group, String item);
/**
 * request a file name from the user
 *
 * @param group    choice group
 * @param item     item from group identifying message
 *
 * @return String  name of file chosen, or else null if canceled
 */
String file(String group, String item);
/**
 * push a parameter list to the UI
 *
 * @param item     name of parameter list
 * @param value    parameter list
 *
 */
public void setParams(String item, Hashtable value);
/**
 * get a parameter list from the UI
 *
 * @param item     name of parameter list
 *

```

```

    * @return Hashtable parameter list
    */
    Hashtable getParams(String item);
    /**
    * make component visible (or not)
    *
    * @param key    name of component
    * @param on     visible or not
    */
    void setVisible(String key, boolean on);
    /**
    * make component visible (or not)
    *
    * @param i      selects the i'th parallel display
    * @param key    name of component
    * @param on     visible or not
    */
    void setVisible(int i, String key, boolean on);
    /**
    * request parallel displays from Fashion
    */
    void split(int i);
}

```

In general, the methods `getUI`, `setUI` and `dispose` connect the Fashion object to its UI object. The methods `nextContainer` and `split` organize the allocation of resources among Fashion object. The methods `file`, `confirm`, and `message` communicate with the user. The method `tree` is used to present interface items/widgets. The method `setVisible` request the presentation of the user interface components controlled by this Fashion object.

Code for UI is shown next. The UI is the glue that binds together a Map with logic and a Fashion with knowledge of presentation. In addition, a `SafeBundle` supplies descriptive information interpreted by the Map and Fashion.

```

    * A UI has some obvious members:
    /**
    * current Map object

```

```

    */
    Map map;
    /**
     * current Fashion object
     */
    Fashion fashion;
    /**
     * current ResourceBundle with Map/Fashion data
     */
    ResourceBundle description;
    /**
     * parent UI
     */
    UI mother;

    * The construction parameters for a new UI include:
    * @param    mother    the parent of this
    * @param    mapName   name of Map class
    * @param    fashionName name of Fashion class
    * @param    box       Container for presenting UI

```

Pseudo-code for the constructor code is shown below:

1. Use the mapName and fashionName to lookup the ResourceBundle description.
2. Create new map and fashion objects by name

(Java: class.forName(mapName).newInstance())

3. Invoke the setUI method of both the map and fashion objects with this UI as a parameter. That is, each Map and Fashion belong to a UI that provides some communication logic. This method of Map classes can construct subordinate UI objects, for which this UI will be the parent. Other methods allow Maps to register with the parent UI to receive notification of events associated with user interactions. Other methods provide for restarting and termination of the user interface.

Another exemplary Fashion class is used for the typical interface display. The class can instantiate the conventional interface widgets such as buttons, labels, text areas,

and others. When the Map asks that a display be presented (the method tree), this Fashion gets a description of the interface from the ResouceBundle of its UI parent. This description specifies what choices and information are to be displayed for the user.

Each piece of the display is composed of "parts" and each part may itself be an additional collection of parts. The main logic of this class (buildTrees and buildSubTrees) recursively analyze the description of parts.

For each primitive part, its description tells what kind of interface primitive to use (a button, for instance) and specifies parameters for the primitive (the button's label, and what icon should decorate the button when it is pushed, for instance).

For parts with parts, the description tells how to layout the subparts (whether a list of buttons is to be stacked horizontally or vertically, for instance). For each interface primitive that has an action associated with it (pressing a button, for instance) the action event is set to call the treeAction method of the Map of the parent UI.

A class for UI objects with map and fashion members is shown next:

```
/**
 * A UI is the glue that binds together a Map with logic and a Fashion
 * with knowledge of presentation. In addition, a SafeBundle supplies
 * descriptive information interpreted by the Map and Fashion. The Map and
 * the Fashion share the Preferences and MessageLog held by the UI object.
 * <p>
 * This UI class will query the Map for instructions on creating a
 * subordinate UI object, if required. The UI's are linked as
 * <code>parent</code> and <code>daughter</code>.
 * <p>
 * Map and Fashion objects are constructed by the UI by loading and
 * instantiating the proper classes by name via the intelligent cache.
 * The SafeBundle is also supplied by the cache.
 *
 * @version    %I%, %G%
 * @author    Chansler
 */
class UI {
```

```

/**
 * registry for all 'action' buttons created by fashions.
 * entries should be JButtons.
 */
private static final Hashtable actionRegistry = new Hashtable(47);
/**
 * label position in parsed description
 */
protected static final int LABEL = 0;
/**
 * clue position in parsed description
 */
protected static final int CLUE = 1;
/**
 * accessibility position in parsed description
 */
protected static final int ACCESS = 2;
/**
 * parts position in parsed description
 */
protected static final int PARTS = 3;
/**
 * key position in a label
 */
protected static final int KEYPOS = 0;
/**
 * position of remainder of label
 */
protected static final int LABELPOS = 2;
/**
 * syntax terminal for parsing descriptions
 */
protected static final char PLANSEP = ';';
/**
 * key in resource bundle for the root of choice tree
 */
protected static final String ROOT = "ROOT";
/**
 * key to look up window title
 */
protected static final String DESC = "DESC";
/**
 * key to look up application title
 */
protected static final String TITLE = "TITLE";
/**

```

```

    * text of clue in description resource that identifies a on/off item
    */
protected static final String OPTION_CLUE = "option";
/**
    * key suffix to look up dialog text
    */
protected static final String TEXT_PART = "-text";
/**
    * key to look up query window title
    */
protected static final String QUERY = "QUERY";
/**
    * key to look up message window title
    */
protected static final String MESG = "MESG";

/*
    * These instance variables have package scope so that they can be
    * conveniently shared by the Map and the Fashion. If protected,
    *   • <code>get</code>, but not <code>set</code> methods would be
    *   • required.
    */
/**
    * name of Map
    */
String mapName;
/**
    * name of Fashion
    */
String fashionName;
/**
    * current Preferences
    */
Preferences preferences;
/**
    * current diagnostic message facility
    */
MessageLog messageLog;
/**
    * current connection to server
    */
IntelligentCache intelligentCache;
/**
    * current Container for UI presentation
    */
Container container;

```

```

/**
 * current Map object
 */
Map map;
/**
 * current Fashion object
 */
Fashion fashion;
/**
 * current ResourceBundle with Map/Fashion data
 */
SafeBundle description;
/**
 * parent UI
 */
UI mother;
/** global resources */
static Globals globals;
static Geoff geoff;

/**
 * queue of listener for UIEvents from this object
 */
protected EventListenerList listenerList = new EventListenerList();
/**
 * general constructor parameters for all imported objects
 *
 * @param    mother    the parent of this
 * @param    mapName   name of Map class
 * @param    fashionName name of Fashion class
 * @param    pref      current Preferences
 * @param    log       diagnostic message facility
 * @param    cache     current intelligent cache
 * @param    box       Container for presenting UI
 *
 * @return    UI
 */
UI(UI mother,
    String mapName,
    String fashionName,
    Preferences pref,
    MessageLog log,
    IntelligentCache cache,
    Container box){
    this.mother = mother;
    if (mother == null) {

```

```

    globals = new Globals();
    globals.setPromiscuous("UI.globals");
    geoff = new Geoff(log, pref, cache);
}
this.fashionName = (fashionName == null) ? "FashionNull":fashionName;
this.mapName = (mapName == null) ? "MapNull":mapName;
preferences = pref;
messageLog = log;
intelligentCache = cache;
container = box;
Trace.traceTemp("new UI(" + mapName + ", " + fashionName + ")");

description = cache.newBundle(this.mapName, this.fashionName);
listenToWindow();

map = (Map)cache.newObject(Map.class, this.mapName);
fashion = (Fashion)cache.newObject(Fashion.class, this.fashionName);

if (fashion != null){
    fashion.setUI(this);
} else {
    messageLog.panic(this, "failed to create fashion " + this.fashionName);
    return;
}

if (map != null){
    map.setUI(this);
} else {
    messageLog.panic(this, "failed to create map " + this.mapName);
    return;
}
}
/**
 * request to quit (from Map); fires Quit UIEvent
 *
 * @param why    identification of request source
 */
protected void quit(Object why){
    messageLog.log(this, "Good bye! Thanks for all the fish! " + why);
    fireUIEvent(new UIEvent(this, UIEvent.UIQuit));
}
/**
 * Request to restart (from Map); fires Restart UIEvent.
 * Restart is conditioned upon there being a real need. Generally
 * requested when personality parameters are suspected of having been
 * changed.

```

```

*
* @param why identification of request source
*/
protected void restart(Object why){
    messageLog.log(this, "Restarting UI because of " + why);
    fireUIEvent(new UIEvent(this, UIEvent.UIRestart));
}
/**
* listen to window event; called if container is a JFrame
*/
protected void listenToWindow(){
    if ( ! (container instanceof JFrame))
        return;
    JFrame jframe = (JFrame)container;

    WindowListener listen = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {quit(this);}
    };
    jframe.addWindowListener(listen);
}
/**
* make UI go away
*/
public void dispose(){
    if (map != null) map.dispose();
    if (fashion != null) fashion.dispose();
}

/**
* Adds a UIListener to the UI.
*/
public void addUIListener(UIListener listener){
    listenerList.add(UIListener.class, listener);
}

/**
* Removes a UIListener from the UI.
*/
public void removeUIListener(UIListener listener) {
    listenerList.remove(UIListener.class, listener);
}

/*
* Notify all listeners that have registered interest for
* notification on this event type.
* (Borrowed from swing implementation, without lazy events. -RC)

```

```

* @see EventListenerList
*/
protected void fireEvent(UIEvent event) {
    // Guaranteed to return a non-null array
    Object[] listeners = listenerList.getListenerList();
    // Process the listeners last to first, notifying
    // those that are interested in this event
    for (int i = listeners.length-2; i>=0; i-=2) {
        if (listeners[i]==UIListener.class) {
            ((UIListener)listeners[i+1]).UIAction(event);
        }
    }
}
/**
 * Construct a combination record for use as an actionCommand.
 * (must necessarily be a String and not a simple record class)
 */
String actionCommand(String parent, String part, String clue){
    return parent + UI.PLANSEP + part + UI.PLANSEP + clue;
}

/**
 * Define a shared notion of "selected" for different preference items.
 *
 * @param id typically the actionCommand for node
 *
 * @return boolean
 */
protected boolean selected(String id){
    String[] details = StringUtils.string2Array(id, UI.PLANSEP);
    return (OPTION_CLUE.equals(details[2]))
        ? preferences.getBooleanValue(details[1], false)
        : preferences.sameas(details[0], details[1]);
}

void registerAction(String actionID, JButton action) {
    if (actionRegistry.put(actionID, action) != null)
        messageLog.panic(this, "Action replaced in registry!");
}

void setActionEnabled(String actionID, boolean enabled) {
    JButton jb = (JButton) actionRegistry.get(actionID);
    if (jb == null) {
        messageLog.panic(this, "(dis/en)abling non-existent action!");
    } else {
        jb.setEnabled(enabled);
    }
}

```

```

    }
  }
}

```

An exemplary top-level map called “MapZero” that implements the top level menus is discussed next. This implementation of the Map interface is the Map at the root of the UI tree.

MapZero functions:

- *
- * select Locale from list
- * select Fashion from list
- * select Map from list
- * select Look'n'Feel from list
- * select whether to use ToolTips
- * present Help and About message dialogs
- * load/save Preferences
- * exit UI
- *
- * To change personality parameters, the Map saves the new selection in the Preferences, and requests that the UI restart. If the Help or About dialogs are to be presented, the Map makes a direct request of the Fashion. The Map makes direct requests to turn ToolTips on and off. If application exit is requested, the Map notifies the UI.
- * <p>
- * This Map relies on an appropriate resourceBundle.

When the parent UI constructs the MapZero object and calls the setUI method of this class. That method constructs the one subordinate UI: (A more complex Map would do this for each subordinate UI.)

1. Get the name of the next Map from the ResourceBundle of the parent UI.
2. Get the name of the next Fashion from the ResourceBundle of the parent UI.
3. Ask the Fashion of this UI for the display resouces (nextContainer) to be used for the new UI.
4. Construct the new UI.
5. Register for event notifications from the new UI.

The business logic of this map manages the application preferences. The treeAction method of this class just sets the appropriate parameters.